

本周周报(7.9-7.15):

解聪

本周工作:

1. 本周重点还是集中在交易地图的工作方面。
这部分的工作主要包括交易数据的动态效果处理。
在经过上周的融入铁路轨迹工作后，基本效果如下图所示：

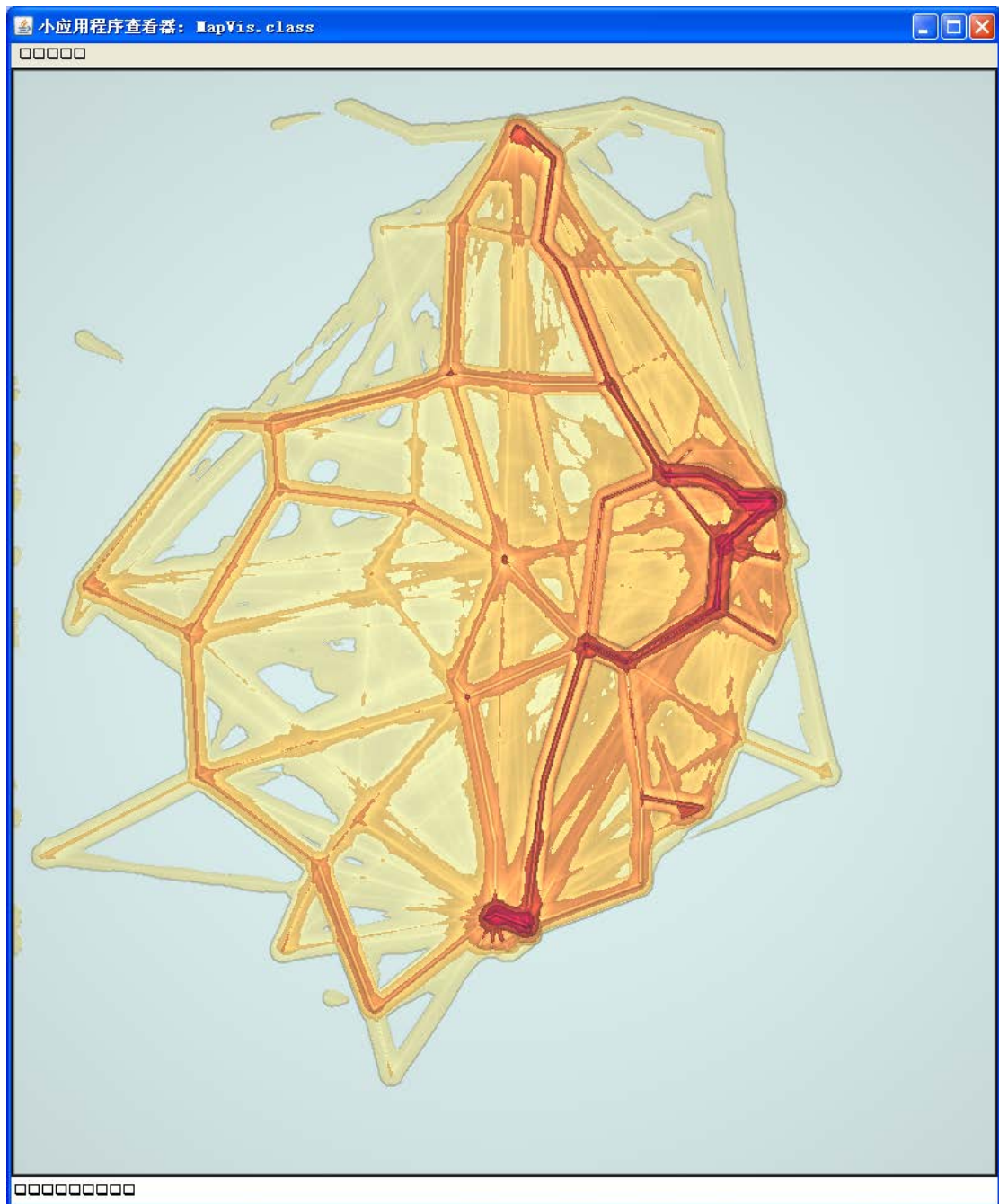


图 1.程序使用 Processing 绘制，配色由深到浅表示密度由大变小。

首先尝试在 java 平台上利用 processing 实现动态的效果。预期的效果是彗星划过的效果，或者是血液在血管里面流动的效果

半球状粒子动态效果的实现

为实现动态效果，需要对源程序的数据结构进行修改。

数据结构如下：

为每一笔交易创建一个对象，成员变量是交易的各种属性，包括交易的起始终止点，交易的类型：飞机或铁路，交易的路径（使用最短路算法预先计算并保存）。

新建一个活化的交易队列。遍历统计出的每一笔交易，如果交易在当前的时间开始进行，则将交易添加到活化队列中。如果交易结束，将交易从活化队列中删除。每绘制一帧，就对活化交易队列更新一遍。

建立两个 Density Map：

Density Map A 保存背景的密度图信息，就是没有粒子运动效果的密度图。Density Map A 在一定时间窗口内统计交易数据，并使用 KDE 计算出的密度图。（目前这个是静态的，即只是统计了一个小时的数据。下一步的工作就是要将背景变为动态数据）。

Density Map B 保存的是单笔交易导致的局部密度变化信息。当每笔交易划过地图时，其交易路径会所经过的密度图会显示出一定的突起。通过遍历每一笔当前发生的交易，计算并修改 Density Map B。每绘制一帧，Density Map B 更新一次。

为了提高效率，每一笔交易划过时，只改变交易路径所在的局部像素点。

算法步骤如下：

1. 对于密度场中的每一个点，首先更新 Density Map B。对于交易路径周围的点，对其密度图对应的值给予一定增量。
2. 对密度图中每个更新点的八领域采用 Phong 模型重新计算其光照。
3. 对于光照有更新的点，重新计算其 RGB 值并且更新到屏幕上。

实现效果如下：

由于分辨率问题，再加上点和背景采用相同的配色方案，图片效果看的不是很清楚。

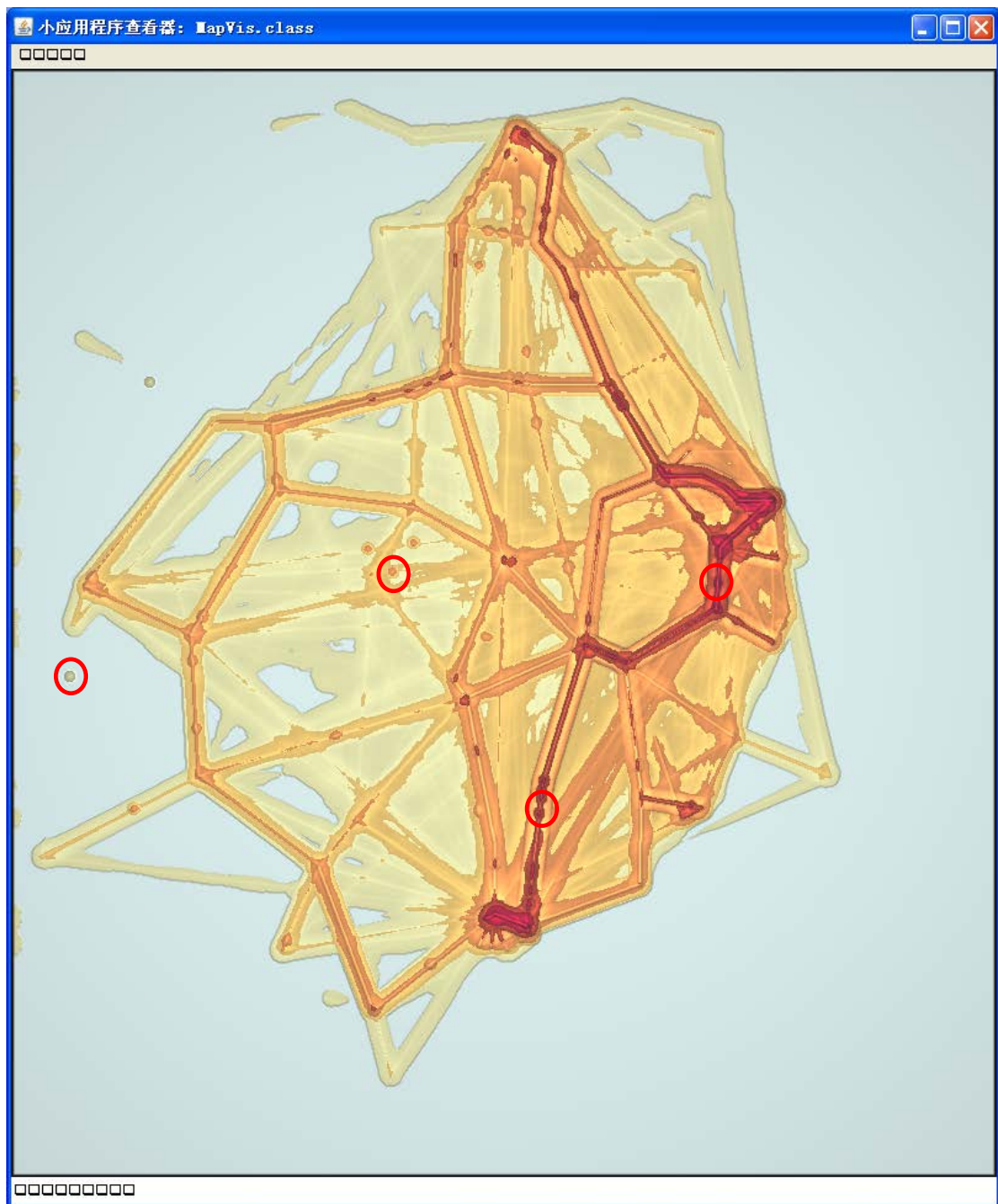


图 2.动态交易的截图效果。使用半球状的突起表示一笔交易正在进行。球的半径为 6px。球的配色方案与背景配色方案一致，颜色的深到浅表示“高度”的高到低。

粒子拖尾效果的实现:

由于小球的运动效果并没有达到预期的水平。因此，接下来又实现了小球的拖尾效果，即小球划过的地方会留下类似于“尾巴”的痕迹。而随着交易的划过，旧的痕迹会慢慢消失，前方的交易路径又会留下新的痕迹。

为了实现这种效果，需要保留前 N 帧的图像并插值生成。由于配色方案由密度图的值映射生成，所以而在不同帧之间直接对颜色插值的方式显然能达到要求。因此需要结合前 N 帧

的密度图，重新计算当前点的密度值，最后再加光照并绘制。

因此与半球运动效果不同之处在于，本方法保存了前 N 帧的 Density Map。

基本算法如下：

1. 对于密度场中的每一个点，每次对比相邻两张 Density Map 与原始 Density Map A 的区别。
 - 1).如果某点处当前帧密度值与原始密度值一致，而先前帧密度值与原始值不一致，则表示当前交易已经划过该点，此点当前就处于“尾巴”的位置；
 - 2).如果某点处当前帧密度值与原始帧密度值不一致，先前帧密度值与原始帧密度值一致，则表示有交易正处在此点位置。
 - 3).如果某点处当前帧密度值、先前帧密度值与原始帧密度值一致，表明无交易路径经过。
2. 计算出当前帧中，拖尾效果导致的 Density Map 值的变化。
3. 使用 Phong 模型重新计算光照。
4. 在屏幕上更新 RGB 值。

每次绘制更新时，交易粒子的移动步长为 4 个像素，则拖尾效果“尾巴”的长度最长为 $4*4=16$ 个像素。

如果要改变拖尾的长度，可采用一下两种方法：

- 1).加长移动的步长，这样会使得移动变得更快。可以认为是增加了移动的速度；
- 2).增加所保存的先前帧的 Density Map 数量。由于保留了更多帧，可以计算出更详细的历史路径。但是这样会占用更多的内存。

本次实现中保留了前 4 帧的 Density Map，下图显示的是是实现拖尾的效果。

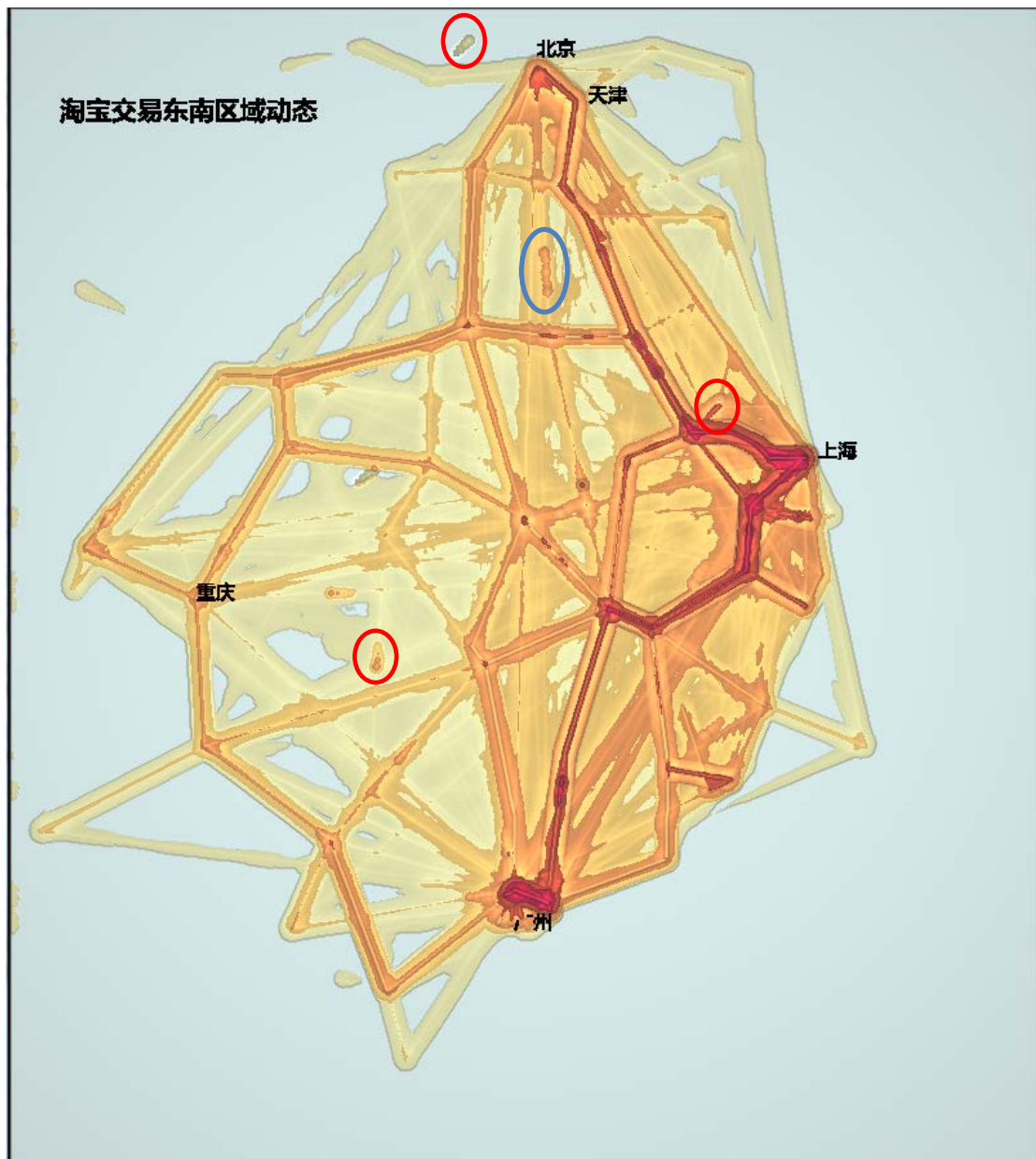


图 3. 添加了拖尾效果。球的半径是 6px，实现中保留了前 4 帧的 Density Map。其中蓝色的圈表示有多笔交易在地图某处相遇混合的效果

对与以上方法，我增加了一些必要的文字说明，比如在地图上将相应城市名称标明出来。并录制了一段视频，视频的效果（仅仅是录制软件的效果）比周六的好一点。

视频地址：<ftp://10.76.0.157/Incoming/Users/xiecong/taobao7.14.mp4>

现有的问题：

1. 由于尚未采用并行编程，且 Java 平台效率不高。下一步拟采用 CUDA 实现并行编程。在经过与吴斐然师兄讨论后，了解到 Java 平台下可以使用 CUDA，即 JCUDA。这样就避免了平台移植到 VS 上所浪费的时间。下周先熟悉一下 JCUDA 的编程。
2. 之前提到进行放大/缩小交互时，KDE 是否重新计算的问题。我参考了“Interactive Visualization of Streaming Data with Kernel Density Estimation”。

本文中在进行放大/缩小交互时，采用了一系列自适应的方法。先修改了 `kernel` 的尺度以及带宽，再利用新的 `kernel` 对密度场重新计算。

如下图所示：

最左边的图显示的是 KDE 的某处细节，作者采用了较小的核。中间一幅图是用户放大操作后显示的图像，其计算密度的时候又换了一个较大的核。

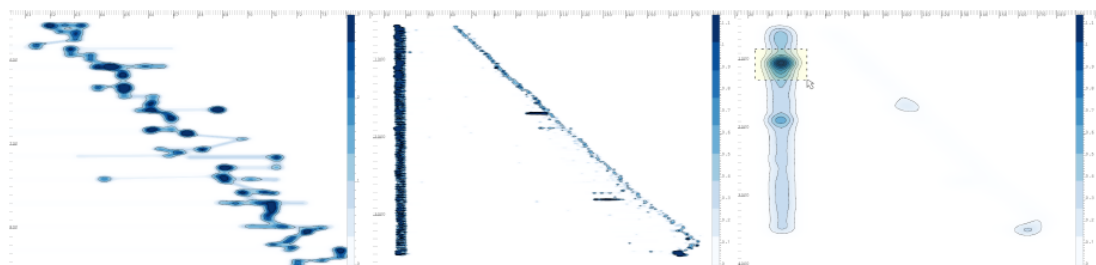


Figure 10: Three line kernel density estimates, showing the distribution of time over depth in a drilling hole (wellbore) and hook-load, the weight of the entire drill string. The leftmost image is a detailed view, with a small kernel, showing the curve with varying tons on the hook, used e.g., to calculate friction. The user then zooms out, and goes to an overview mode with a large kernel, on right, and selects an overwhelming time density, integrates and finds that over an hour was nonproductive at this depth.

我觉得这种方法对于淘宝交易数据是有合理之处的。比如：在现有的交易数据看来，长三角地区是比较密集的，地图上呈深红色。如下图所示：

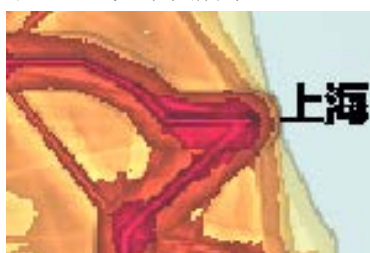


图 5：长三角区域密度图。该区域交易密集，如果简单放大的话，会忽略掉该区域的许多细节信息。因此需要重新选取半径较小核函数反应细尺度的信息。

假设用户放大，单独显示长三角地区的密度图。如果仍然采用原先的核函数，用户在地图上会看到全部都是红色的区域，分不清细节信息。

因此，可以将核半径缩小。用户可以探索在长三角区域中哪些地区是相对更繁忙（比如杭州），哪些地区相对不那么繁忙（比如位于长三角区域的一些小镇）。

无论交互如何实现，还是需要先实现快速、并行处理 KDE 的方法。在这基础上，再处理进一步的问题，比如实时的数据流的 KDE 计算，或者是解决放大 / 缩小的交互问题。

下一步工作：

1. 尝试动态更新密度图。需要提升 KDE 计算的性能。因此尝试使用 JCUDA 在现有平台上进行并行计算密度图。
2. 优化与改进动画效果。
3. 改进进出交易的可视化效果，比如采用纹理动画的方式。

2. DataV.js 组件库的工作。

周二参加了可视化组件库的讨论，确定了组件发布的时间是 7 月底。并且规划了未来两三周

组件库的工作。

本周由于设计人员尚未规划好统一的设计方案，本周现就各人的组件做性能优化，包括针对不同浏览器制定的不同显示策略以及算法实现中的优化。

另外，由于李逢博的离职，她所负责的弦图的组件先暂时交给我负责。

下周工作：

1. 交易地图下一步的工作包括：实时 KDE 计算的实现与现有可视化的改进。
2. 按照设计人员讨论后的方案修改组件库的交互与外观。
3. 教材整理。